IN THE SPECIFICATION:

Please replace the title with the following amended title:

Method for Transparent, Location-Independent, Remote Procedure Calls in a ~~Hetrogeneous~~ Heterogeneous Network Environment

Please replace paragraph [0001] with the following amended paragraph:

[0001] The present invention relates to execution of a function and/or procedure call on a remote node of a network environment.

Please replace paragraph [0023] with the following amended paragraph:

[0023] The present invention generally overcomes the deficiencies of conventional methods by allowing a function call and the respective parameters associated with the function call to be available to more than just non-server nodes. Therefore, for example, a function (x) and its associated data resident on node 1 of a network may be read and or generally accessed by another node on the network. In this configuration, when function (x) is called, then the function itself may be configured to determine whether the functional operation would be most efficiently executed locally or on another remote node. If function (x) is best suited to be executed on another node, then function (x) may call a routing function. The routing function may be configured to package the parameters associated with function (x) from the stack and transmit the packaged parameters to the node best suited to execute the function. Once the parameters are received on the remote node, function (x) may be invoked on the remote node and processed thereon using a replicate stack. When the function exits and/or completes, then the remote node packages the return data and transmits this data back to the originating node. The originating node, which is ~~note~~ node 1 in the present exemplary embodiment, may then copy the return data into the appropriate places. Alternatively, if the local node, *i.e.* node 1, is capable of executing the function, then the function may simply be executed on the local node without involving any routing and/or transmitting of data to a remote node.

Please replace paragraph [0026] with the following amended paragraph:

[0026] Therefore, referring to the exemplary configuration of Figure 11, when a function is called on Node 3, it may be determined that the specific function is best executed via Application A. Therefore, it may be determined that the function should be transferred to either Nodes 1, 4, or 7. Assuming that Node 1 is selected, then the function and its related data may be transparently transmitted to Node 1 for execution. Upon executing the function, Node [[7]] 1 may transmit the resulting function data back to Node 3. Node 3 may receive the resulting function data and process the data as if it were executed locally, all of which is generally transparent to the user on the local originating node. Thus an application/function can essentially run on any node in the system and can be moved around without affecting the processing results of the application/function. For example, application A can be run on node 1 and have most of the commands travel across the network. But since the API is generally consistent on all nodes and each function knows where to route the commands to get the operations executed. Thus, Application A could simply be loaded and executed on node 2 without a recompile or any changes and therefore execute much faster. Further, it should also be noted that if node 2 is running a different OS microprocessor type or other data type, a recompile may be necessary, however, the recompile would not affect the operation of the API.

Please replace paragraph [0037] with the following amended paragraph:

[0037] Based on the above noted data types expressed in DTSTRUCT, the ROUTE function, as briefly discussed above, essentially operates to flatten each of the variable arguments specified in the call to the ROUTE function into a buffer. The flattening process, which will be further discussed herein, generally includes compressing and/or selecting portions of data that are to be transferred across the network to the remote node so that transmission latency delays may be minimized. The DTSTRUCT and function service identifier may also be included in the buffer contents, so that the opposing node will be able to decode and/or unpackage the data and call the appropriate function on the opposing node. If cache keys are specified in DTSTRUCT, then the data value of these keys may then be checked to see if the remote reply is resident in cache memory. If the data is in cache memory, then the ROUTE function may forgo the routing operation by unflattening the cached response and immediately exit. If the data is not resident cache, or alternatively, if there are no keyed parameters, then the buffer may be transmitted across the network to the remote node by the ROUTE function. The remote node receives the command and buffer contents and decodes the buffer using the DTSTRUCT passed thereto in the buffer contents. While the remote node unflattens the buffer contents, it also generates a call stack that generally mirrors, in structure, the original call stack on the local originating node.

Please replace paragraph [0049] with the following amended paragraph:

[0049] Once the data required to execute FUNCXYZ() is unflattened on the remote node at step 209, the method continues to step 210. At step 210 the method ~~user~~ uses a service identifier passed in the buffer to locate the address for FUNCXYZ() on the remote node. The lookup function pointer may be generally configured to determine where FUNCXYZ() exists on the remote node, as the function address is almost certainly different from the address indicated by the local originating node. Therefore, the lookup function pointer may use a registry method, as is generally known in the art, in order to determine the location of FUNCXYZ() on the remote node.